VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE TÉMAT Z MLUVENÉ ŘEČI

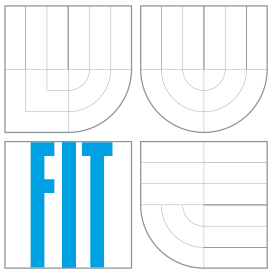
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK ŠKEŘÍK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE TÉMAT Z MLUVENÉ ŘEČI

TOPIC DETECTION FROM SPOKEN SPEECH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK ŠKEŘÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. SCHWARZ PETR, Ph.D.

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá detekcí témat z mluvené řeči. Zpracováním a převodem mluvené řeči na text se zabývá první část práce. Samotný problém detekce témat je řešen dvěma odlišnými přístupy - strojovým učením a expertním přístupen kladení velmi přesného dotazu na dokument. Obě metody jsou testovány nad sadou dat, statisticky vyhodnoceny a porovnány.

Abstract

This thesis is about topic detection from spoken speech. The first part of the thesis deals with speech transcription to text. The thesis describes two different solutions of the topic detection - a machine learning based solution and an expert solution that composes a very precise query describing the document topic. Both methods are tested on a set of recordings and compared.

Klíčová slova

Detekce témat, přesně kladený dotaz, strojové učení, mluvená řeč, řečové technologie, přepis řeči, Phonexia.

Keywords

Topic detection, exactly asked query, spoken speech, speech technologies, speech transcription, Phonexia.

Citace

Zdeněk Škeřík: Detekce témat z mluvené řeči, bakalářská práce, Brno, FIT VUT v Brně, 2015

Detekce témat z mluvené řeči

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Schwarze, Ph.D.

.....
Zdeněk Škeřík
13. května 2015

Poděkování

Rád bych poděkoval mému vedoucímu a společnosti Phonexia za příležitost dělat smysluplnou práci a možnost naučit se nových věcí.

Obsah

Kapitola 1

Úvod

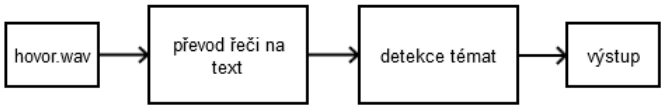
Bakalářská práce se zabývá detekcí témat z mluvené řeči. O nástroj, který by umožňoval detekci témat z mluvené řeči, je mezi klienty společnosti *Phonexia s.r.o.*[1] (dále jen Phonexia) zájem, a pro nové či potencionální zákazníky je tento nástroj zajímavé lákadlo. Některé konkurenční společnosti již nabízejí svá řešení, a proto se Phonexia rozhodla přijít se svým vlastním. Toto řešení je předmětem této práce.

Všeobecně je detekce témat nad informacemi v digitální podobě poměrně mladé odvětví. Tím jak roste úložný prostor, dochází k přesunu dat z klasických nosičů na ty digitální. Potřeba tyto data efektivně spravovat, klasifikovat a vyhledávat v nich přináší řadu problémů. Zvládnutím tohoto úkolu je odměna v podobě znalostí získaných z uložených dat, protože hodnota dat, v kterých neumíme vyhledávat, je nulová.

Nezbytným krokem, který předchází samotnou detekci témat, je přepis mluvené řeči na text. Phonexia se zabývá prací s mluvenou řečí a disponuje technologiemi pro přepis řeči na text, konkrétně dvěmi. Obě jsou v této práci využity. Jejich princip fungování je popsán v kapitole 2.

Na úvod bych chtěl specifikovat termín *mluvená řeč*. V kontextu této práce jsou pod tímto pojmem zastoupeny audio nahrávky, které jsou pořizovány z drtivé části kontaktními centry. Z toho vyplývá, že detekce témat bude probíhat zpětně a nikoliv v době hovoru, jedná se o tzv. offline verzi. Charakter nahrávek je takový, že v mluvené řeči probíhá dialog mezi operátorem a klientem, avšak obecně analýza a detekce témat může být prováděna nad jakoukoliv mluvenou řečí. Jediná nutná podmínka je, aby nahrávka byla v dostatečně dobré kvalitě, a to z důvodu přesnosti řečových technologií. Výstup z těchto technologií má přímí vliv na úspěšnost detekce témat.

Práce přichází s dvěmi implementacemi tohoto úkolu. V zásadě se jedná o dva zcela odlišné přístupy - *detekce témat založená na přesně kladeném dotazu* 5 a *detekce témat založená na strojovém učení* 6. Obě varianty mají své výhody a nevýhody, které budou diskutovány v příslušných kapitolách. Oba přístupy budou statisticky vyhodnoceny nad sadou testovacích dat. Charakter dat a testovací scénář je popsán v kapitole 7.



Obrázek 1.1: Obecné schéma systému pro detekci témat z mluvené řeči

[1] http://www.phonexia.com

Detekce témat založená na přesně kladeném dotazu bude implementována a plně integrována do produktu *SPAS*[2] společnosti Phonexia. Vyjma logické vrstvy bude dále nutné také implementovat databázovou vrstvu pro ukládání výsledků a uživatelské grafické rozhraní pro snadné a intuitivní ovládání. Uživatelskému rozhraní přikládám velkou důležitost, a proto jemu věnována samostatná kapitola 4.

V závěru práce 8 jsou zhodnoceny výsledky projektu, jeho využití a přínos v praxi. Jsou zde shrnuty jednotlivé přístupy a jejich vlastnosti. Také je zde diskutován další vývoj a směrování. Práce si klade za cíl nezůstat jen u detekce témat, která by sloužila pouze ke klasifikaci hovorů, ale jít dále. Detekce témat bude sloužit jako základ pro hlubší analýzu nahrávek, jako například sofistikovanější kontrolu dialogu, získávání dat, apod.

[2] http://www.spas-solution.com/

Kapitola 2

Řečové technologie

2.1 Phonexia s.r.o.

Jak bylo zmíněno v úvodu, projekt je psán z iniciativy společnosti Phonexia. Nadcházejících pár řádků je proto věnováno této společnosti.

Společnost byla založena v roce 2006 na půdě Vysokého učení technického fakulty Informačních technologií. Hlavním portfoliem produktů jsou řečové technologie, které úspěšně převádí z akademické sféry do komerční. Mezi hlavní přednosti firmy patří úzká spolupráce s Fakultou informačních technologií, konkrétně s řečovou skupinou *BUT Speech@FIT* z ústavu počítačové grafiky a multimédií. Tato spolupráce zaručuje přístup k nejnovějším postupům zpracování mluvené řeči. V současné době firma nabízí několik technologií, které jsou adaptované pro několik jazyků. To umožňuje firmě navazovat obchodní kontakty po celém světě.

2.2 Úvod do použitých technologií

Práce využívá pro zpracování řeči knihovnu *Brno Speech Core (BSCORE)* s rozhraním *Brno Speech Application Interface (BSAPI)* [1]. Knihovna nabízí širokou škálu algoritmů pro práci s řečí. Avšak v této práci jsou použity pouze dvě technologie - přepis řeči na text a vyhledávání klíčových slov v řeči. Knihovna BSCORE a rozhraní BSAPI jsou napsány v jazyce C. Pro účely této práce jsem využil rozhraní *BSAPI wrapper*, které celou knihovnu zapouzdřuje a zpřístupňuje i pro jiné programovací jazyky, v našem případě pro jazyk Java.

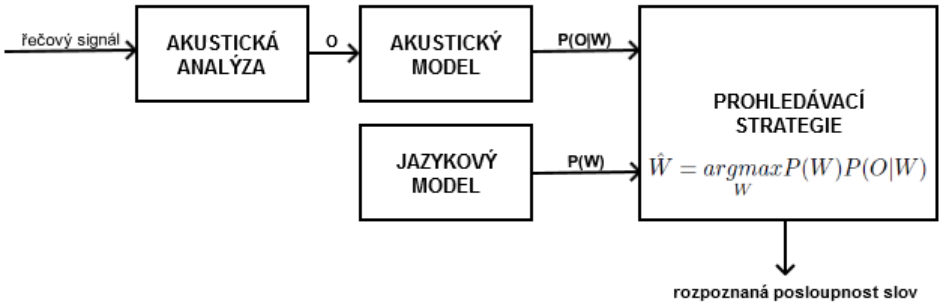
Obě použité technologie jsou si z principu fungování velmi podobné, jsou založeny na **statistickém** přístupu rozpoznávání řeči. Jejich princip bude probrán v nadcházejících kapitolách. Práce si neklade za cíl jít do detailu, ale pouze obecně nastínit jejich princip fungování. Samostatný detailní popis by vydal na celou práci. Teorie fungování řečových technologií vychází z těchto zdrojů [4], [6], [7], [11].

První metodu, kterou zde uvedu, je přepis řeči. V kapitole 2.3 bude popsán jeho princip fungování. Kapitola o vyhledávání klíčových slov 2.4 je pouze doplňující, budou zde zmíněny rozdíly oproti přepisu řeči. Každá z metod má své výhody a nevýhody, které vyplývají z jejich principu fungování.

2.3 LVCSR

***L**arge **V**ocabulary **C**ontinuous **S**peech **R**ecognition* je přepis řeči s velkým slovníkem postavený na statistickém klasifikátoru slov. Velikost slovníku záleží na daném jazyce, jedná se v řádu o statitisíce slov. V projektu je použita offline verze přepisu řeči využívající český slovník.

Celý problém přepisu řeči můžeme dekomponovat na několik samostatných podproblémů, které lze řešit nezávisle, viz následující obrázek 2.1.



Obrázek 2.1: Systém rozpoznávání řeči dekomponovaný do bloků podle řešených úloh (Obr. 5.2 strana 199 z knihy [11])

Předpokládejme, že $W = \{w_1, w_2, w_N\}$ je posloupnost N slov a nechť $O = \{o_1, o_2, o_T\}$ je akustická informace, tj. posloupnost vektorů příznaků, odvozená z řečového signálu, z níž se řečový dekodér pokouší rozpoznat, které slova byla vyslovena. Cílem je nalézt posloupnost slov $\hat{W}$, které maximalizuje podmíněnou pravděpodobnost $P(W|O)$, tj. nejpravděpodobnější posloupnost slov pro danou akustickou informaci O.[11]

Matematicky tento vztah můžeme vyjádřit jako

$$\hat{W} = \underset{W}{argmax} P(W|O)$$

po aplikaci Bayesova pravidla dostáváme vztah

$$\hat{W} = \underset{W}{argmax} P(W)P(O|W)$$

ze kterého vyplývá, že stanovení nejlepší posloupnosti slov k danému akustickému signálu lze řešit pomocí dvou samostatných pravděpodobností, které mohou být modelovány a trénovány nezávisle.

Akustickou informaci O získáme **akustickou analýzou** nahrávky, pravděpodobnost $P(O|W)$ získáme z **akustického modelu** a pravděpodobnost $P(W)$ z **jazykového modelu**.

2.3.1 Akustická analýza

Hlavním úkolem akustické analýzy je převod vstupního analogového signálu na signál digitální. Signál se převede z času spojitého do času diskrétního, dojde k tzv. vzorkování, kdy signál je rozdělen na velmi krátké úseky. Hodnoty v těchto úsecích se zaokrouhlují k nejbližší hladině, která je reprezentována číselnou hodnotou. Tímto získáme posloupnost

krátkých úseků signálu reprezentovaných číselnými hodnotami. Tento krok nemusíme řešit, jelikož vstupem řečového dekodéru jsou již nahrávky v digitální podobně.

Dalším krokem je z nahrávky vybrat relevantní data. Jelikož signál obsahuje velmi mnoho informací, které kromě samotných promluv popisují například barvu hlasu, rychlost, šum v pozadí, atd., je potřeba vybrat taková, které nám umožní následné rozpoznávání slov tzv. **příznaky**. Základem analýzy řečového signálu bývá **krátkodobá analýza**, která signál rozdělí na velmi krátké úseky, tzv. **mikrosegmenty**. Typická délka mikrosegmentu je v řádech desítek milisekund a zhruba odpovídá jednomu stavu, ve kterém se nachází artikulačního ústrojí mluvícího člověka. Každý mikrosegment je pak reprezentován vektorem příznaků o. Výstupem akustické analýzy je posloupnost vektorů příznaků $O = \{o_1, o_2, o_T\}$.

2.3.2 Akustický model

Funkcí akustického modelu je získat co nejlepší odhad podmíněné pravděpodobnosti $P(W|O)$ pro vstupní akustickou posloupnost O. Modelování celých slov je alespoň v tuto chvíli nereálné a to z důvodu jejich obrovského počtu. Také modifikace takové modelu by nebyla jednoduchá. Musíme tedy modelovat menší jednotky než jsou slova. Takovou jednotkou jsou **fonémy**. Foném je elementární zvuková jednotka, která umožňuje v jazyce odlišit různé významové jednotky. Například: **v**láda, **k**láda. Z příkladu je jasně vidět, že změnou jednoho fonému dostane slovo zcela jiný význam. Uvažujeme-li, že při řeči se v každém okamžiku nachází hlasové ustrojí v jednom z konečného počtu artikulačních stavů, které se navenek jeví jako fonémy, tak tuto skutečnost můžeme modelovat. Pro modelování se využívá **skrytých Markovových modelů**. Akustický model je soubor natrénovaných skrytých Markovových modelů. Dobře natrénovaný model musí být

- přesný - schopný rozlišit slova, která znějí podobně, ale jejich sémantický význam je odlišný
- flexibilní - charakter řeči může být zcela jiný od trénovaných dat, např.: artikulace, tempo či akustické pozadí

Modelování fonémů přináší hned několik výhod. Pro přidání nového slova do řečového dekodéru není hned nutné přidávat a trénovat nový model, protože toto slovo se skládá z již namodelovaných fonémů. Další důležitá vlastnost takového akustického modelu je, že každý jazyk má omezenou množinu fonémů. Jedná se v řádu o desítky fonémů, to je v porovnání s desítkami tisíc slov velmi velký rozdíl. Důsledkem toho je, že nám stačí modelovat méně modelů a zároveň trénování takového to modelu vyžaduje méně trénovacích dat.

2.3.2.1 Skrytý Markovův model

Skrytý Markovův model je stochastický automat o konečném počtu stavů. V našem případě model představuje jeden foném. Pro modelování řeči se používají tzv. levo-pravé modely, které umožňují pouze přechod do vyšších stavů nebo setrvání ve stavu aktuálním. Při procesu modelování jsou generovány dvě svázané časové posloupnosti náhodných proměnných - **podpůrný Markovův řetězec** a **řetězec vektorů příznaků**.

Řetězec vektorů příznaků pozorování $O = \{o_1, o_2, o_T\}$ reprezentuje spektrální charakter jednotlivých úseků řečového signálu - mikrosegmentů. Ke všem spektrálním vzorům jsou vytvořeny náhodné funkce popisující pravděpodobnost podobnosti mezi mikrosegmenty a stavy automatu.

Podpůrný Markovův řetězec je posloupnost konečného počtu stavů, které reprezentují jednotlivé fonémy. Řetězec mění stavy na základě své matice pravděpodobností přechodu. Posloupnost těchto modelů nám udává konkretní slova či celé věty.

Znalost ocenění jednotlivých pravděpodobností přechodu získáme z množiny trénovacích dat. Obecně lze říci, že čím je množina dat větší a různorodější, tím je akustický model robustnější.

Na závěr této podkapitoly bych chtěl zmínit, že v dnešní době se nemodelují samostatné fonémy. A to z toho důvodu, že jednotlivé fonémy jsou při řeči ovlivňovány fonémy okolními, tedy jejími sousedy, kteří tvoří kontext daného fonému. Toto okolí do značné míry ovlivňuje výslovnost aktuálního fonému. Proto uvažujeme jak levý, tak i pravý kontext fonému. Takové jednotce říkáme *trifon*. V důsledku toho se navyšuje počet stavů modelu, ale výsledkem je přesnější rozpoznávání slov.

2.3.3 Jazykový model

Jazykový model poskytuje bloku prohledávací strategie odhad pravděpodobnosti $P(W)$ pro libovolnou posloupnost slov. Každý jazyk má vyjma slovní zásoby svá specifika a zákonitosti, podle kterých se řídí větná skladba. Právě znalosti popisující stavbu a chování jazyka jsou obsaženy v jazykovém modelu, který tím udává omezení pro daný jazyk. Omezení jsou obecně dvojího druhu - **deterministická** a **pravděpodobnostní**.

Příkladem deterministického omezení je, že řečový dekodér nerozpoznává slova, která nemá uložena ve slovníku.

Pravděpodobnostní omezení většinou udávají s jakou pravděpodobností může nastat náhodná posloupnost N-slov. V praxi se většinou užívá bigramových (posloupnost dvou slov) a nebo trigramových modelů (tři slova). Dobrý jazykový model pro spontánní řeč musí mít oceněnu pravděpodobnost pro libovolnou posloupnost N-slov, protože každá kombinace slov je přípustná.

V této práci je použit obecný jazykový model, který obsahuje více než 300 000 slov. Tento model by měl pokrýt velkou část mluvené řeči. Avšak není výjimkou, že chceme přepis řeči použít pro konkrétní oblast, která je specifická například svoji slovní zásobou či slovními konstrukcemi. V takových to případech se jazykový model upravuje pro danou oblast nasazení. Obecně pak řečový dekodér s takto upraveným modelem dosahuje lepších výsledků.

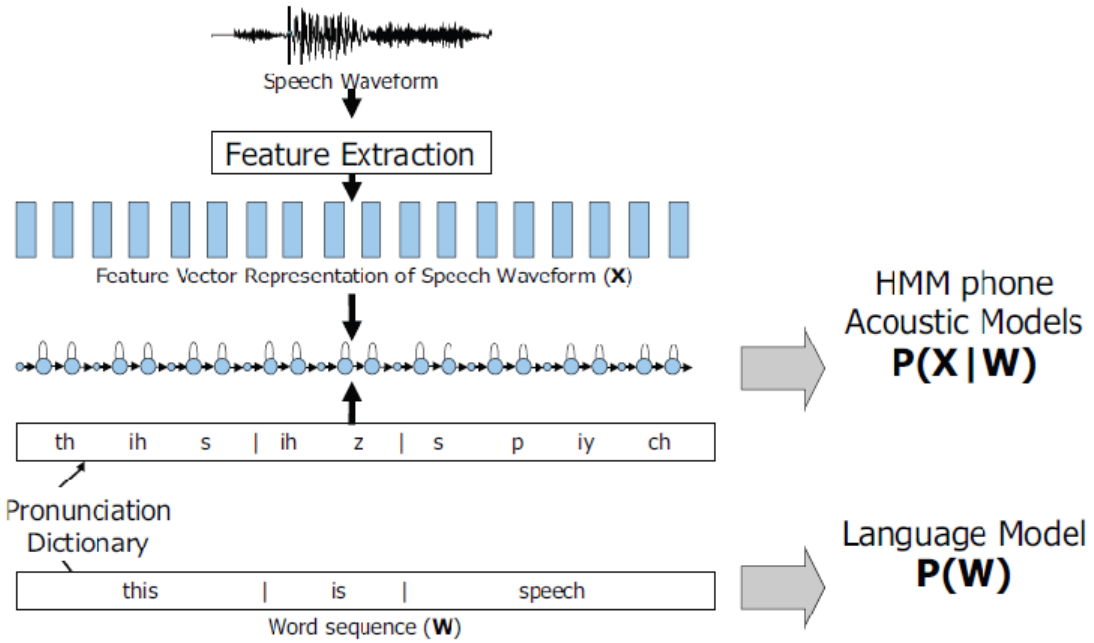
2.3.4 Prohledávací strategie

Následující obrázek 2.2 ještě jednou ilustruje princip přepisu řeči, kde *feature* je příznak, *HMM* je skrytý Markovův model a veličina X vyjadřuje to samé jako O. Na obrázku vidíme dvě výstupní pravděpodobnosti z jazykového a akustického modelu, které tvoří vstup pro prohledávací strategii, kterou dále budeme označovat jako **dekodér**.

Cílem dekodéru je nalézt výstupní posloupnost slov pro vstupní akustický signál. Pro tento úkol se využívá **konečných váhových převodníků** (dále jen **WFST** - ***W**eighted **F**inite **S**tate **T**ransducer*) [7]. WFST se využívají pro kompozici modelů představující daný jazyk. Seznam modelů reprezentuje zkratka **HCLG**. Dekodér pak pomocí HCLG sítě přepisuje fonémy na slova.

Vzorec z kapitoly 2.3

$$\hat{W} = \underset{W}{argmax} P(W)P(O|W)$$



Obrázek 2.2: Ilustrace činnosti přepisu řeči. (Obrázek z 2. strany prezentace [7])

můžeme přepsat a vyřešit jako kompozici následujících váhovaných konečných převodníků

$$\hat{W} = H \circ C \circ L \circ G$$

kde

- **H** je váhovaný konečný převodník převádějící řetězce kontextově závislých fonémů na stavové řetězce
- **C** je váhovaný konečný převodník převádějící fonetický řetězec na jeho kontextovou variantu
- **L** je váhovaný konečný převodník převádějící slova na jejich fonetický přepis
- **G** je konečný automat reprezentující ngramový jazykový model

Dekodér nabízí několik typů výstupních dat.

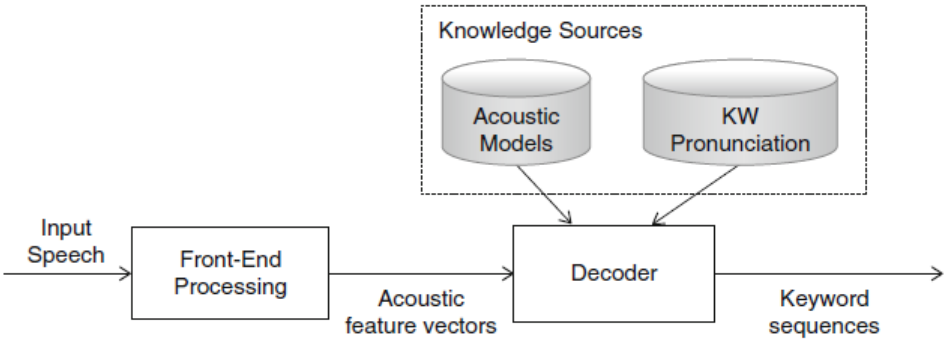
1. **one-best** (nejlepší hypotéza)
2. **lattice** (acyklický dopředný graf reprezentující paralelní hypotézy vět) Výstup z lattice je dále možné upravit na:
   (a) **confusion network** (acyklický dopředný graf reprezentující paralelní hypotézy slov)
   (b) **n-best** (n prvních nejlepších vět)

2.4 KWS

***K**ey**W**ord **S**potting*, v češtině rozpoznávání klíčových slov, je technologie pro vyhledávání slov nebo frází v řeči. Existuje několik přístupů, stručně zde nastíním ten, který implementuje společnost Phonexia.

Jedná se o tzv. **accoustic KWS**, jehož schéma fungování je znázorněno na obrázku 2.3. Stejně jako LVCSR je technologie KWS založena na statistické rozpoznávací síti. Při vyhledávání potřebujeme dopředu specifikovat jaká slova či fráze chceme vyhledávat. Tyto slova jsou pak převedena podle své výslovnosti na posloupnost fonémů a následně na rozpoznávací síť. Akustický model zde plní stejnou úlohu jako v LVCSR.

Dekodér pak pomocí Viterbi algoritmu[1] vyhledává v nahrávce klíčová slova. Rozpoznávací síť obsahuje kromě modelu klíčových slov také model libovolné věty tvořené fonémovou smyčkou. Porovnáním pravděpodobnosti vystupující z modelu některého klíčového slova a modelu libovolné věty získáme skóre klíčového slova. Pro nadetekovaná klíčová slova detektor vrací datovou strukturu popsanou v kapitole 2.5.



Obrázek 2.3: Schéma akustického KWS. (Obrázek 3 z 2. kapitoly z knihy [9])

2.5 Výstup řečových technologií

Jak bylo zmíněno výše, řečové technologie používám skrze rozhraní *BSAPI wrapper*, které implementuje API v jazyce Java. Princip použití je jednoduchý, požadované technologii předám nahrávku ke zpracování a ta v průběhu vrací callbackem jednotlivá rozpoznaná slova. Výstup pro technologie LVCSR a KWS je totožný, výstupem je list objektů typu *TransOneWord*, jehož struktura je následující:

- **word** - rozpoznávané slovo
- **confidence** - pravděpodobnost se kterou slovo bylo detekováno
- **start** - čas začátku slova v nahrávce
- **end** - čas konce slova v nahrávce
- **channel** - kanál, ve kterém slovo bylo detekováno

[1] http://en.wikipedia.org/wiki/Viterbi_algorithm

Kapitola 3

SPAS

Část této práce je řešena v rámci projektu **SPAS** (**SP**eech **A**nalytic**S**). Tato aplikace je primárně určena pro analýzu nahrávek v kontaktních centrech. Denně se pořídí desítky tisíc minut záznamu (záleží na velikosti kontaktního centra), který není nikdy zkontrolován, v podstatě data v něm obsažená nenaleznou dalšího využití. Na tuto skutečnost reagovala Phonexia tím, že začala vyvíjet tento nástroj.

SPAS slouží pro analýzu a správu nahrávek. Je primárně určen pro kontaktní centra, ale obecně ho lze využít všude tam, kde je potřeba analyzovat nahrávky. SPAS je webová aplikace napsána v jazyce Java EE. Nasazená aplikace běží u zákazníka na serveru a uživatelé ji používají skrz webový prohlížeč. Ve SPASu využíváme řečové technologie, nad kterými stavíme další logiku. V současné době můžu SPAS rozdělit na dva funkcionálně odlišné celky - hodnocení operátora a analýza nahrávek.

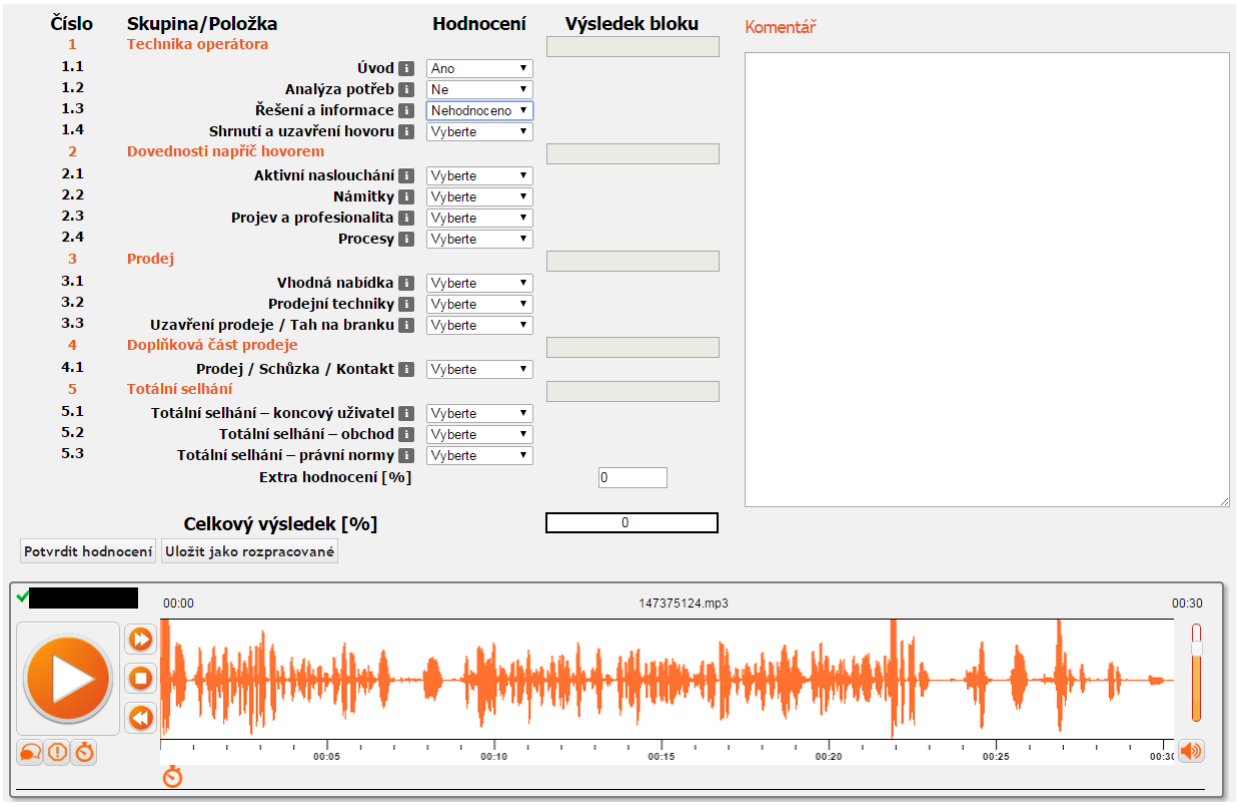
3.1 Hodnocení operátora

Tato funkcionalita je používána pro každodenní kontrolu a hodnocení operátora za účelem zlepšení kvality a služeb. Kontrola je prováděna manuálně a z toho vyplývá, že počet zkontrolovaných hovorů je minimální (v řádech jednotek procent). Takto byly a stále jsou hovory kontrolovány v drtivé části kontaktních center. Kontrola probíhá tak, že se pro danou kampaň vytvoří hodnoticí formulář, podle kterého pak supervizor hodnotí poslouchanou nahrávku.

Funkcionalita hodnocení operátora byla doimplementována za účelem ucelenosti celé aplikace, a to z důvodu, aby vše důležité pro pracovníky a kontrolory kvality (supervizory) bylo na jednom místě.

3.2 Analýza nahrávek

Analýza nahrávek přináší automatizované zpracování **všech** pořízených nahrávek. Na této části aplikace se nejvíce pracuje. Jak bylo řečeno v úvodu této kapitoly, nahrávky se analyzují pomocí řečových technologií a nad výsledky se provádí další vyhodnocení. V současné době využíváme technologie verifikace operátora (SID), statistiku nahrávky (TAE), detekci klíčových slov (KWS) a přepis (LVCSR). Poslední dvě zmíněné využívám pro detekci témat. Nyní aplikace nabízí možnost detekce a jednoduché analýzy místa v nahrávce, která je řešena pomocí KWS. Příkladem použití může být kontrola povinných údajů v nahrávce ať

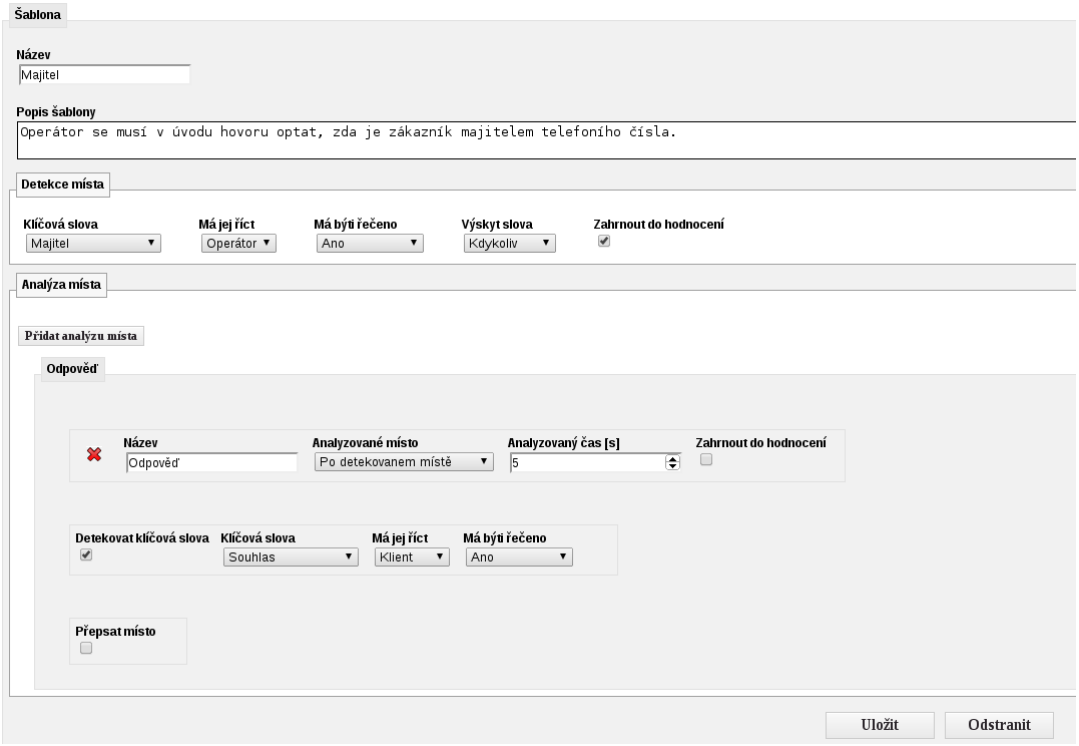


Obrázek 3.1: Hodnoticí formulář v aplikaci SPAS

už podle legislativy (upozornění na to, že hovor je monitorovaný) nebo požadavků kontaktního centra (povinný pozdrav, dotaz na majitele telefonu a následná odpověď volaného). Účel takové analýzy je pak v zásadě dvojí. Podchycení nešvarů jednotlivých operátorů za účelem zlepšení kvality hovoru. Druhým aspektem je pak získání užitečných dat z hovoru. Takto navržená detekce a analýza z dat je však nedostatečná a zároveň tu padl požadavek na detekci témat. Spojení těchto okolností nás vede k vytvoření lepšího nástroje, který by zvládl komplexnější analýzu hovorů, jehož součástí bude i detekce témat.

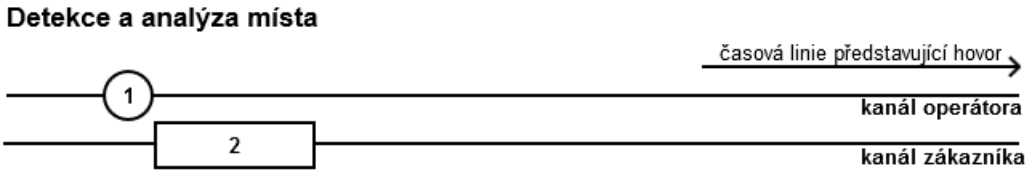
Obrázek 3.2 ukazuje na příkladu původní grafické rozhraní pro definici analýzy hovoru. Tato funkcionalita dovoluje definovat místo (klíčové slovo, fráze), které se v hovoru musí a nebo naopak nesmí objevit. V případě splnění této podmínky pak přichází analýza místa. Většinou analyzované místo nějak souvisí s místem detekovaným. K detekovanému místu se může vázat nula až N analyzovaných míst. Nula proto, že někdy nám stačí v nahrávce pouze detekovat nějakou informaci a nic víc nás nezajímá (příkladem může být detekce samotného pozdravu). Obrázek zobrazuje konkrétní definici, jejíž účelem je zjistit, zda operátor položil otázku, jestli je zákazník majitelem tohoto telefoního čísla. V případě, že tak učinil, analyzujeme zákazníkovu odpověď.

Z obrázku je vidět nastavení. V boxu pro *detekci místa* říkáme, detekuj otázku na majitele, detekuj ji na kanálu operátora a kdekoliv v hovoru. Pokud tato podmínka je splněna, tak analyzuj místo, které je definované v boxu *analýza místa*. V analýze místa říkáme, že chceme detekovat kladnou odpověď (ano) na kanálu zákazníka, odpověď hledej v intervalu 5 sekund po detekovaném místě. Obrázek 3.3 tento scénář ukazuje schématicky. Kruh označen číslem 1 představuje detekci místa (otázku na majitele) a obdélník reprezentuje analýzu



Obrázek 3.2: Původní analýza nahrávek v aplikaci SPAS.

místa (odpověď).



Obrázek 3.3: Schématický příklad detekce na otázku majitele a následná analýza odpovědi.

Na tomto příkladu je vidět celý princip dosavadní analýzy hovoru. Výsledky získané analýzou pak můžeme filtrovat. V návaznosti na předchozí příklad si můžeme například vyfiltrovat všechny hovory, kde nezazněla otázka na majitele nebo kde odpověď byla negativní. Tyto výstupy analýzy můžeme využít pro další práci, získané znalosti využijeme ať už pro lepší trénování operátorů, či získání různých statistik.

Také je ovšem jasné, že tato analýza má svá omezení. Můžeme si ji představit jako stromovou strukturu o maximální výšce dva, kde kořen stromu představuje detekci místa a jeho listy jsou analýzy místa. Taková to analýza nám ovšem nedovoluje analyzovat komplexnější scénáře a proto je nutné implementovat nový, obecnější analyzátor, který toto zvládne a jeho součástí bude i detekce témat.

3.3 Specifikace nového řečového analyzátoru

Řečový analyzátor bude plně integrován do aplikace SPAS. Kromě samotné detekce témat musí umět nástroj detekovat a následně analyzovat části hovoru a získávat data, tzv. *data mining*. Analyzátor musí mít přehledné uživatelské rozhraní jak pro definování jednotlivých případů užití, tak pro prezentaci výstupu.

3.4 Motivace

Zvládnutím tohoto úkolu získá uživatel nástroj, který mu poskytne další užitečné informace k informacím stávajícím. V dnešní době je velký zájem o extrakci dat z kanálů, kde „tečou“ informace v obrovském množství, jedná se o tzv. *big data*. A právě žádná jiná než automatizovaná analýza tak velkého objemu dat není možná.

V neposlední řadě pak jedna zakázka byla více než motivující. Požadavek byl extrakce dat (rodné číslo a telefonní čísla, na která byla v hovoru uzavřena smlouva) z prodejních hovorů jednoho nejmenovaného virtuálního operátora. Z množiny hovorů bylo nejdříve nutné vybrat pouze ty, kde byl uzavřen prodej alespoň na jedno telefonní číslo. Pak bylo nutné vybrané nahrávky přepsat pomocí LVCSR. V přepsaných nahrávkách detekovat místa, kde by se mohla vyskytovat požadovaná čísla, z těchto míst extrahovat čísla (ve slovní podobě) a ta převést do číselné podoby. Nakonec nahrávky přejmenovat ve tvaru *ID-RČ-TEL1-TELn.WAV*. Toto jsme byli schopni udělat, ale nikoliv plně automatizovaně.

3.5 Návrh řešení

Celou implementaci a začlenění analyzátoru do aplikace SPAS můžeme rozdělit na několik podproblémů.

- Logická vrstva představující samotný analyzátor jehož možností bude detekce témat, viz kapitola 5.
- Návrh databázové vrstvy, kde budou uloženy jak analyzované scénáře tak výsledky.
- Návrh grafického uživatelského rozhraní pro zadávání scénářů, viz kapitola 4.
- Návrh výstupu z analyzátoru.

Kapitola 4

Grafické uživatelské rozhraní

Tato kapitola se věnuje prezentační vrstvě skrze kterou je zadávána celá detekce témat, potažmo celý analyzátor hovorů. Kapitolu zařazuji před samotnou detekci témat 5 kvůli její důležitosti, a to z důvodu, že na grafických ukázkách si čtenář lépe představí celou problematiku, ať spojenou s celým vývojem, zadáváním a vyhodnocováním scénářů pro detekci témat. Scénář zde reprezentuje přesně kladený dotaz.

4.1 Návrh řešení

Grafické uživatelské rozhraní pro analýzu hovorů je implementováno jako samostatná stránka v rámci aplikace SPAS. Stránka umožňuje přehledně definovat scénáře pro analýzu hovorů, která zahrnuje detekci témat. Z důvodu přehlednosti a intuitivního ovládání je celý scénář zobrazován v grafické podobě. Toto zobrazení umožňuje komplexně vidět celou vztahovou síť jednotlivých částí analýzy a přehledně se v ní orientovat a upravovat či dále ji budovat.

4.2 Použité technologie

SPAS je webová aplikace, z tohoto důvodu se odvíjejí použité technologie. Samotná aplikace je vyvíjena v jazyce **Java EE**[1]. Kromě základních frameworků nezbytných pro fungování celé aplikace, jako jsou **Maven**[2] a **Spring**[3], je pro nás důležitý **Apache Wicket**[4] framework, která má na starosti prezentační vrstvu. Další technologie, které jsem použil pro implementaci GUI jsou **Javascript**, potažmo **JQuery**[5].

4.2.1 Apache Wicket

Apache Wicket je framework pro implementaci prezentační vrstvy webové stránky v jazyce Java. Je to komponentově orientovaný framework s vysokou abstrakcí nad protokolem HTML. Framework je vyvíjen od roku 2004, nyní je v aplikaci použita verze 6.

Jádrem celého frameworku je práce s modely, které slouží pro propojení POJO (javovský) objektů s HTML komponentami (formuláře, tlačítka, rolovací nabídky, ...). Propojení

[1] http://www.oracle.com/technetwork/java/javaee/overview/index.html
[2] http://maven.apache.org/
[3] https://spring.io/
[4] https://wicket.apache.org/
[5] http://jquery.com/

javovské části s HTML komponentou je realizováno skrz atribut *wicket:id*. Pro lepší pochopení uvádím následující příklad, který ukazuje implementaci zaškrtávacího tlačítka, jehož hodnota se promítne do javovského objektu.

```
//Java
Boolean checkbox = false;
CheckBox chb = new CheckBox("checkboxId", new PropertyModel <Boolean>(this, "checkbox"));

//HTML
<input type="checkbox"wicket:id="checkboxId"/>
```

Z příkladu vidíme, že v Java kódu nejdříve definuje proměnnou typu boolean *checkbox*, dále pak objekt *Checkbox* z knihovny frameworku Wicket. Jako první parametr udáváme id (wicket:id), které nám propojí objekt s HTML komponentou. Druhý parametr je **model** typu *Boolean*, atribut *this*, který říká, že komponentu svážeme s proměnnou *checkbox*, která se nachází v tomto kontextu (kontextu této stránky). V HTML kódu definujeme komponentu *input* typu *checkbox* a oproti klasické definici HTML přidáváme atribut *wicket:id*.

Mezi další velké výhody tohoto frameworku je podpora technologie **AJAX**, která nám dovoluje psát vlastní implementace událostí jednotlivých HTML komponent. Tato technologie je podpořena jednak samotnými ajaxovými komponentami nebo je možné přímo implementovat ajaxové chování. V návaznosti na předchozí ukázku uvádím příklad implementace ajaxového chování. Při zmeně stavu zaškrtávacího tlačítka se invokuje metoda *onUpdate*, ve které můžeme implementovat příslušné chování, atd.

```
//Java
...
CheckBox chb = new CheckBox("checkboxId", new PropertyModel <Boolean>(this, "checkbox"));
chb.add(new AjaxFormComponentUpdatingBehavior(AttributeName.ON_CHANGE){
    @Override
    protected void onUpdate(AjaxRequestTarget target) {
        // do some work
    }
});
```

4.2.2 Javascript

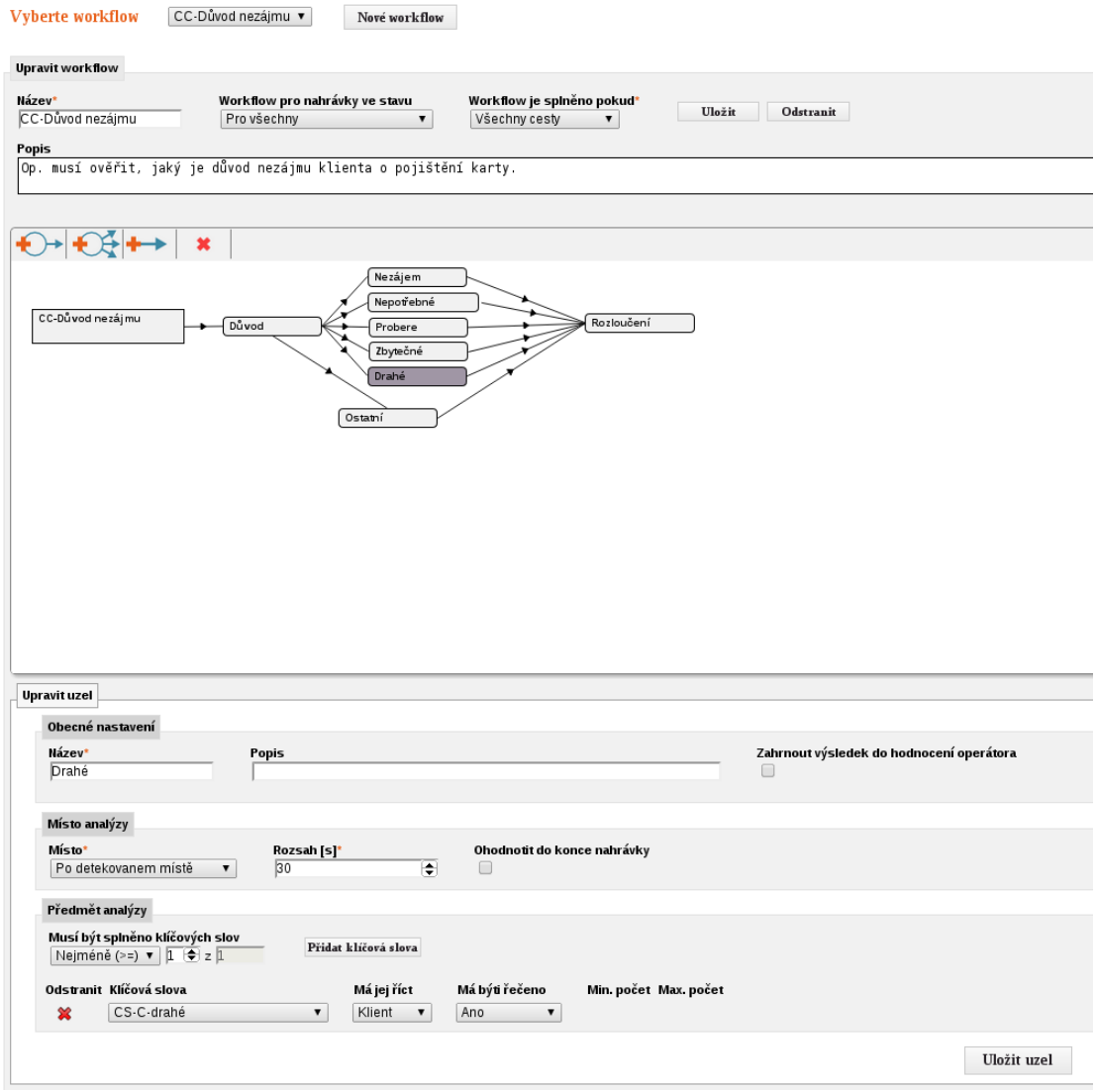
Použití javascriptu bylo nezbytné pro implementaci grafického zobrazení scénáře, které by bylo pomocí frameworku Wicket nerealizovatelné. Samotné implementaci předcházela volba vhodné javascriptové knihovny, která podporuje práci s diagramy. Nakonec jsem zvolil knihovnu **jsPlumb**[6], která poskytuje to, co potřebuji, dále má dobrého průvodce a přehledné API. Dále jsem si práci usnadnil použitím **jQuery**, které již aplikace SPAS využívá, a které práci oproti obyčejnému javascriptu ulehčuje, navíc samotný kód působí přehledněji.

4.3 Implementace

Na úvod kapitoly uvádím pro přesnost terminologii, která je použita v dalším textu. Analyzovaný scénář je znázorněn pomocí **sítě**. Síť reprezentuje *acyklický orientovaný graf*. Síť je složena z **uzlů**, kde uzel představuje jeden krok analýzy. V uzlu je definována příslušná akce, například detekce určitého slova či fráze. V analogii s původní analýzou hovoru popsanou v kapitole 3.2, by detekce místa představovala jeden uzel a analýza místa uzel další.

[6] http://www.jsplumb.org/demo/flowchart/dom.html

Uzly jsou propojeny **přechody**, které představují nikoliv časovou, ale logickou následnost. Přechod je orientovaná hrana grafu.



Obrázek 4.1: Webová stránka pro definici nové analýzy nahrávky.

Na obrázku 4.1 vidíme stránku, která implementuje grafické rozhraní pro zadávání analýz. Tato webová stránka je reprezentována třídou *TopicDetectionPage.java*. Stránku lze rozdělit do tří částí. Ve vrchní části je nabídka umožňující přepínání mezi jednotlivými sítěmi a tlačítko pro vytvoření nové sítě. Prostřední části dominuje panel, ve kterém je graficky znázorněna celá síť. Pod tímto panelem je formulář, ve kterém definujeme jednotlivé uzly sítě.

Koncept celé stránky je takový, že celá je implementována v jazyce Java za použití komponent z knihovny frameworku Wicket. Panel zobrazující síť je implementován v javascriptu a do stránky injektován při jejím vytvoření. Pro správné fungování celé stránky, potažmo zajištění validace a konzistence celé sítě (grafická síť musí reflektovat její logickou podobu uloženou v databázi) bylo nutné implementovat následující kroky:

1. Inicializace grafického panelu implementovaného v javascriptu.

2. Komunikace a synchronizace grafické sítě se zbytkem stránky, která je implementována v Javě.

3. Validace sítě.

   (a) Validace sítě jako celku.

   (b) Validace jednotlivých uzlů.

4.3.1 Grafický panel

Jak bylo napsáno výše, grafický panel vizualizuje podobu sítě pomocí javascriptové knihovny **jsPlumb**. Grafický panel, dále jen panel, je javascriptový objekt, který se stará o:

1. Vykreslování sítě.

2. Tvorbu, mazání a propojování uzlů.

3. Při změně stavu (viz odrážka výše) komunikaci se stránkou, která implementuje logiku.

Zdrojový kód je umístěn v samostatném souboru *input_graphic_chart.js*. Panel je implementován jako objekt, který poskytuje veřejné rozhraní pro komunikaci a soukromé metody pro správu sítě. Objektového chování je zde dosaženo zapouzdřením funkcí, viz návrhový vzor **Model**, podkapitola jQuery na stránce [10].

4.3.1.1 Struktura objektu

Objekt panel je vytvářen přes konstruktor, který jako parametry přebírá ID HTML komponenty, do které bude injektován (ve které bude zobrazovat síť), síť k zobrazení, jejichž reprezentace je uložena v jazyce *JSON*, lokalizaci pro zobrazení v daném jazyce a seznam callbacků (url adresy) pro komunikaci se zbytkem stránky.

Panel je rozdělen na dvě části. Ve vrchní části je menu, pod ním se nachází již samotné plátno zobrazující síť.

Veřejné metody

Panel implementuje sadu veřejných funkcí pro komunikaci **stránka → panel**. Komunikační rozhraní implementuje následující funkce:

1. **newNet()** Vytvoří novou síť.

2. **saveNet()** Uloží celou síť ve formátu JSON a jako řetězec ji callbackem zašle stránce.

3. **loadNet(netInJSON)** Načte a vykreslí předanou síť.

4. **setNodeName(nodeId, nodeName)** Přejmenuje uzel s daným identifikátorem.

Soukromé metody
Soukromé metody se starají o vykreslování, tvoření a validaci sítě. V důsledku množství metod, je zde nebudu jednotlivě rozebírat, ale jen shrnu jejich obecný význam a užití.

Topologie sítě je budována čistě v panelu, který je tedy správcem celé sítě. Každý uzel musí nést unikátní identifikátor v síti, podle kterého je možné jej rozpoznat či vyhledat. Jelikož síť je pouze nositelem grafické reprezentace, tak je v uzlu uloženo pouze jeho jméno. Všechna data představující definici uzlu jsou zadávána ve formuláři implementovaném na webové stránce v jazyce Java, více v kapitole 4.3.2.

Pro opačnou komunikaci (**panel** → **stránka**) jsou panelu předány *url adresy callbacků*. Tyto callbacky zachytává webová stránka a implementuje potřebné chování, více viz 4.3.2.2.

Panel se dále stará o validaci sítě jako celku. Síť nemůže být tvořena ledabyle, je potřeba stanovit jistá pravidla a omezení. Některá vychází ze samotného typu sítě, která představuje acyklický orientovaný graf. Hrany jsou orientovány a tok může procházet pouze ve směru hran. Dále uvádím výčet omezení:

1. Nelze propojit dva stejné uzly více hranami.
2. Každý uzel musí mít minimálně jednu vstupní hranu.
3. Více uzlů může vstupovat do stejného uzlu.
4. Není možné tvořit smyčky.

Samotný uzel je tvořen pomocí jQuery. Jedná se o objekt, kterému je přiřazen unikátní identifikátor v rámci sítě. Uzel nese pouze informaci o svém jménu. Uzlu je ještě přidán css styl, vlastnost pro možnost "táhnutí"a nakonec je předán panelu pro správu.

O tvorbu hran se již stará samotná knihovna jsPlumb a to konkrétně funkce *jsPlumb.connect*, která jako parametr přebírá objekt, který specifikuje dvojici uzlů k spojení a další atributy popisující vzhled.

4.3.2 Webová stránka

4.3.2.1 Inicializace grafického panelu

Grafický panel je v javovské části zapouzdřen a používán skrze třídu *InputGraphicChart.java*. Tato třída se stará o vytvoření jedné instance objektu, která se vytvoří při inicializaci celé stránky. Pro vytváření objektu a injektování do stránky používám metodu frameworku Wicket jménem *renderHead()*. Přepsání této metody nám umožní vložit potřebné javascriptové knihovny do hlavičky stránky při jejím vytváření. Následující příklad demonstruje inicializaci objektu.

```
public String getInitializeJs() {
    String resultJS = "var chart;";
    resultJS += "jsPlumb.ready(function () {";
    resultJS += "chart = new InputGraphicChart ('id', netInJSON, callbackUrl_1, callbackUrl_n);"
    resultJS += "});";
    return resultJS;
}
```

```
@Overriden
public void renderHead(IHeaderResponse response) {
    super.renderHead(response);
    // css soubor pro grafický panel
    response.render(CssHeaderItem.forUrl(CssConstants.TOPIC_CHART));
    // javascriptová knihovna jsPlumb
    response.render(JavaScriptHeaderItem.forUrl(JsConstants.JS_PLUMB));
    // javascriptový soubor s grafickým panelem
    response.render(JavaScriptHeaderItem.forUrl(JsConstants.TOPIC_CHART));
    // script pro inicializaci objektu grafický panel, viz výše
    response.render(JavaScriptHeaderItem.forScript(getInitializeJs(), "init"));
}
```

Pro ilustraci zde ještě uvádím použití metod, které nám nabízí rozhraní grafického panelu. Vybral jsem pouze jednu metodu, ostatní se volají analogicky.

```
// načtení a zobrazení nové sítě
public void loadNet(String netInJSON, AjaxRequestTarget target) {
    target.appendJavaScript("chart.loadNet("+ netInJSON + ");");
}
```

4.3.2.2 Komunikace s grafickým panelem

Ovládání panelu bylo popsáno výše, probíhá prostřednictvím objektu definovaném v třídě *InputGraphicChart.java*, který umožňuje volat veřejné metody grafického panelu. Je ovšem nutné zajistit také komunikaci opačnou a to zejména z důvodu, kdy potřebujeme předat informaci o stavu sítě zbytku webové stránky, která s touto informací nadále pracuje. Tato komunikace je nezbytná k udržení konzistence celé sítě. Jak jsem již zmínil dříve, grafická reprezentace sítě musí odpovídat její interní podobě, kterou představují třídy *TopicNet.java* a *TopicNode.java*, jejichž stav je pak uložen v databázi.

Pro komunikaci **objekt → stránka** nám slouží callbacky. Callback v tomto kontextu je url adresa, na kterou panel zasílá zprávy. Webová stránka implementuje chování, které odchytává zprávy na těchto adresách a dále s těmito informacemi pracuje. Stránka implementuje následující callbacky:

Jméno	**Parametry**	**Popis**
activeNodeCallback	id_node	Id aktivního uzlu (uzel, který je označen myší). Zobrazí ve formuláři definici tohoto uzlu.
saveNetCallback	net_definition	Řetězec obsahující definici celé sítě ve formátu json. Uloží síť do databáze.
newNodeCallback	id_parent_node, id_new_node, name_new_node	Vytvoří nový uzel a vloží jej do databáze.
newConnectionCallback	id_source_node, id_target_node	Vytvoří nové spojení mezi dvěma uzly a vloží jej do databáze.
deleteNodeCallback	id_node_to_delete, pole id_source_node	Odstraní uzel a všechna spojení, která s ním souvisela a uloží stav do databáze.
deleteConnectionCallback	id_source_node, id_target_node	Odstraní spojení mezi dvěma uzly a uloží stav do databáze.

Ukázka implementace callbacku v Javě za použití frameworku Wicket a následné volání z javascriptu.

```
// Java
final AbstractDefaultAjaxBehavior callback = new AbstractDefaultAjaxBehavior() {
    private static final long serialVersionUID = 1L;
    @Override
    protected void respond(AjaxRequestTarget target) {
        IRequestParameters parameters = RequestCycle.get().getRequest().getRequestParameters();
        String parameter_value = parameters.getParameterValue("parameter_name").toString();
        // Do some work with parameters
    }
};

// Javascript
function invokeCallback(parameter_value) {
    Wicket.Ajax.get({
        "u": callbackUrl,
        "ep":[
            { "name": "parameter_name", "value": parameter_value }
        ]
    });
}
```

4.3.2.3 Formulář pro definici uzlu

Formulář je implementován pomocí komponent z frameworku *Wicket*. Účelem formuláře je umožnit uživateli definovat či editovat jednotlivé uzly v síti. Definice uzlu se ve formuláři zobrazí po kliknutí na požadovaný uzel v grafické síti, viz *activeNodeCallback*. Formulář je rozdělen do tří hlavních částí.

V první, která je nazvána *Obecné nastavení*, definujeme základní informace o uzlu. Každý uzel má jméno, které by mělo reprezentovat jeho účel. Dále se zde nachází komentář pro bližší specifikaci účelu. Jako poslední můžeme vidět zaškrtávací tlačítko, které udává, zda se výsledek uzlu započítá do hodnocení operátora.

V druhé části formuláře definujeme místo analýzy, to jest v jaké části hovoru budeme daný uzel vyhodnocovat. Nabídka nabízí tři možnosti - před předchozím uzlem, po předchozím uzlu a nebo v celé nahrávce. První dvě možnosti ještě musíme doplnit o časový úsek, v kterém bude analýza probíhat.

Poslední část formuláře specifikuje, co budeme analyzovat. Jedná se o 1 až n klíčových slov. U každého klíčového slova specifikujeme zda má, či nemá být řečeno. Také volíme kanál nahrávky. V případě mono nahrávek kanál nehraje roli. Pokud máme stereo nahrávky, volíme mezi kanálem klienta nebo operátora. Pokud jsou tyto kritéria splněna, pak je daný uzel vyhodnocen jako splněný.

Kapitola 5

Detekce témat založená na přesně kladeném dotazu

Mějme množinu dat, z které se snažíme zjistit charakter a povahu informací a z nich získat znalost o této množině. Tu získáme pouze přesně kladeným dotazem, který je zcela v naši režii. Neexistuje zde žádná pomocná inteligence, která by mohla ovlivnit výsledek. Získáme přesně to, na co jsme položili dotaz. Z toho vyplývá, že klíčovým prvkem tohoto přístupu je člověk, respektive analytik, který sestavuje dotazy.

Pro tento přístup jsem použil dvě řešení. Každé je založeno na jedné řečové technologii. Proč jsem použil obě dvě řešení a jejich porovnání je vysvětleno v následujícím textu.

5.1 Porovnání vlastností KWS a LVCSR

Řešení pomocí KWS obnáší řadu výhod a nevýhod, které pramení z vlastností této technologie. Mějme nahrávku, která je jednokanálová (mono) o délce 1 minuty.

Nahrávka bude technologií KWS zpracována přibližně za 1/10 celkového času nahrávky, technologie LVCSR má přibližnou rychlost rovnající se délce nahrávky. Z toho plyne že KWS je přibližně 10x rychlejší než LVCSR.

Také paměťové nároky nejsou tak znatelné. Vlastním měřením jsem došel k výsledku, že jedna instance KWS pro český jazyk potřebuje přibližně 200 MB operační paměti, zatímco LVCSR spotřebuje 2 GB paměti. Tyto údaje jsou pouze hrubým odhadem, který jsem dostal Linuxovým nástrojem *top*. Ovšem takový odhad nám pro představu stačí. Tím končí výhody KWS, další jsou již jednoznačně na straně LVCSR.

Při použití LVCSR se celá nahrávka zpracuje pouze jednou a tím získáme celý přepis nahrávky, všechna slova, která byla řečena. Zatímco při použití KWS musíme dopředu znát klíčová slova, která chceme vyhledávat. Pokud si v budoucnu uvědomíme, že bychom chtěli hledat slova jiná, musíme celou nahrávku nechat zpracovat znovu. To nám implikuje další nevýhodu, a to takovou, že velká část kontaktních center uchovává nahrávky jen za poslední určité období a to z důvodu jejich obrovského množství.

Poslední velmi důležitou výhodou LVCSR je, že dokáže vracet alternativy rozpoznaných slov.

5.2 Řešení pomocí KWS

Toto řešení bylo implementováno hned z několika důvodů. Tím hlavním byla nutnost počkat na vydání *Elasticsearch verze 1.5* viz následující kapitola 5.3. Rozhodnutí bylo učiněno z důvodu, že první verze detekce témat by měla být vyvinuta co nejdříve. Při práci na této verzi jsem si uvědomil širší souvislosti a odhalil další problémy týkající se tohoto úkolu. Dále mohla být paralelně vyvíjena prezentační vrstva, která díky své vizuální stránce pomohla k lepšímu pochopení a komunikaci s mým nadřízeným. Nakonec se toto řešení stalo finálním.

5.2.1 Návrh řešení

Následující osnova popisuje sadu kroků nezbytných k analýze nahrávek a potažmo detekci témat za využití technologie KWS.

1. definuj klíčová slova pro KWS
2. sestav síť pro analýzu nahrávek v grafickém rozhraní
3. zpracuj nahrávky pomocí technologie KWS
4. vyhodnoť síť nad nahrávkami, ve kterých byla nalezena klíčová slova
   (a) pokud klíčová slova nalezená technologií KWS odpovídají síti, analyzovaný scénář je splněn
   (b) pokud ne, scénář není splněn
5. ulož výsledky do databáze

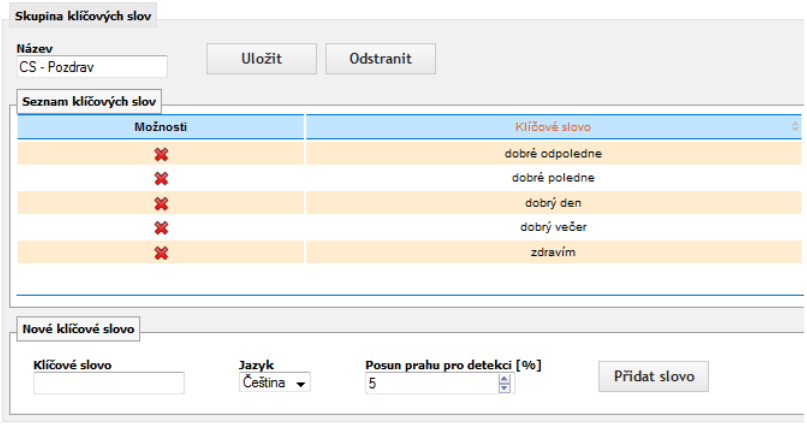
5.2.2 Implementace

Body číslo 1 a 2 z předchozí kapitoly zde přeskočím. Definice klíčových slov je realizovaná v aplikaci SPAS pomocí samostatné stránky. Tato definice je triviální. Na obrázku 5.1 vidíme definici klíčových slov pro pozdrav. Definujeme zde skupinu klíčových slov jménem Pozdrav a seznam pozdravů. Pokud pak zaznělo v nahrávce alespoň jedno slovo z tohoto listu, tak víme, že v nahrávce zazněl pozdrav. Bodu číslo dva je věnována samostatná kapitola 4 této práce.

V této kapitole se budu věnovat hlavně bodům 3 a 4. Ukážu zde princip zpracování nahrávky technologií KWS a následný algoritmus pro analýzu a detekci témat v nahrávce.

5.2.2.1 Zpracování nahrávky technologií KWS

Jak bylo řečeno dříve, pokud chceme použít KWS, musíme dopředu znát slova, která chceme vyhledávat. Právě tyto slova získáme z dříve nadefinované sítě. Následující pseudokód ilustruje celý mechanizmus zpracování.



Obrázek 5.1: Definice skupiny klíčových slov v aplikaci SPAS. Skupiny jsou využívány pro analýzu nahrávek.

```
function processRecordByKWS(Record, Network) {
    KeywordSpotting kws;
    // Projdi síť a z každého uzlu ulož klíčová slova do listu
    List<KeywordCriteria> keywordCriteriaList = getKyewordCriteriaFromNetwork(Network);
    // KWS bude vyhledávat klíčová slova předaná v listu
    kws.initKWS(keywordCriteriaList);

    // KWS hledá v nahrávce klíčová slova podle zadaných kritérií
    List<Keyword> keywordList = kws.processRecord(Record)
    // Vrať klíčová slova nalezená v nahrávce pro další analýzu
    return keywordList;
    });
}
```

5.2.2.2 Analýza nahrávky

V předchozím odstavci jsem ukázal, jak je nahrávka zpracována pomocí technologie KWS. Výstupem KWS je list klíčových slov, který byl v dané nahrávce nalezen. Tento list nám slouží jako vstup pro analýzu nahrávky. Abychom nahrávku vyhodnotili, v zásadě potřebujeme porovnat tento list klíčových slov se sítí, podle které má být nahrávka vyhodnocena.

Základní účel algoritmu je takový, že musí projít celou síť (acyklický orientovaný graf) a vyhodnotit každý uzel. Uzel je splněn, pokud kritéria v něm zadaná odpovídají klíčovým slovům, která byla nalezena v nahrávce technologií KWS. Průchod sítí je implementován pomocí algoritmu **nerekurzivní prohledávání do šířky**. Tento postup ilustruje následující pseudokód.

```
function analyzeRecordByNetwork(record, network, List<Keyword> keywordList) {
    NetworkResult result = new NetworkResult(); // celkové hodnocení sítě
    Queue<TopicNode> queue = new Queue();
    queue.add(network.getRootNode())
    while(!queue.isEmpty()){
        actualNode = queue.poll(); // Vyjmi první uzel z fronty
        // Přídej všechny potomky
        queue.addAll(actualNode.getAllChild());
        // Všechny předchůdci musí být ohodnoceny, jinak uzel zatím nehoďnoť
        if (checkSrcNodesHaveEvaluation(actualNode)){ continue; }
        // Vrať aktuální časovou pozici v nahrávce ve které se vyhodnocování nachází
        // int fromTime = getFromTime(actualNode);
        // Ohodnoť uzel
        evaluateNode(actualNode, keywordList, fromTime);
    }
    // Ulož výsledky do databáze
    insertNetworkResultIntoDB(networkResult);
}
```

Tím jsem ukázal průchod sítí, následující pseudokód znázorňuje vyhodnocení uzlu.

```
function evaluateNode(node, List<Keyword> keywordList, fromTime) {
    NodeResult result = new NodeResult(); // hodnocení uzlu
    Criteria criteria = node.getCriteria(); // kritéria pro tento uzel
    // keywordList obsahuje všechna klíčová slova, je potřeba vybrat pouze ta, která souvisí s tímto uzlem
    List<Keyword> keywordsForThisNode = getKeywordsForNode(node, keywordList, fromTime, criteria)

    if (criteria EQUALS keywordsForThisNode) {
        result.setResult(TRUE);
    } else {
        result.setResult(FALSE);
    }
    // Ulož výsledky do databáze
    insertNodeResultIntoDB(nodeResult);
}
```

Tyto tři pseudokódy ukazují celý princip analýzy hovorů pomocí KWS. Pseudokódy jsou zjednodušeny z důvodu přehlednosti a jednoduché názornosti, celá analýza je pak složitější.

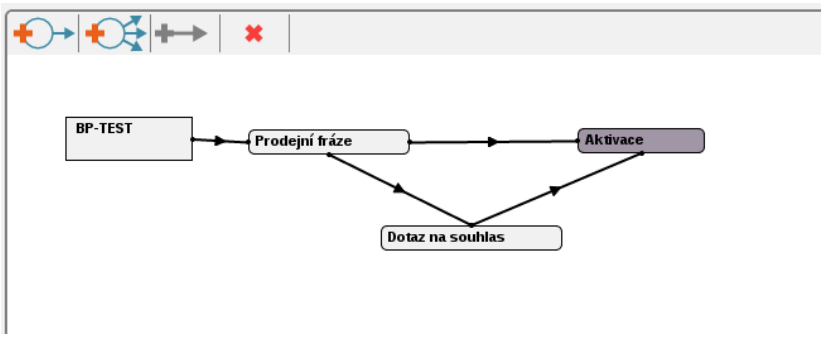
Výstupem analýzy je vyhodnocení pro každý uzel a pro síť jako celek. Tyto výsledky jsou uloženy do databáze pro pozdější využití. Jednak nám udávají výsledky pro jednotlivé hovory, ale dále nám umožňují tvořit dlouhodobé statistiky. Můžeme z nich získávat znalosti, jak si jednotliví operátoři či celé kampaně vedou v čase, jaká místa (uzly) jsou nejvíce kritická a kde je prostor pro zlepšení. Práce s výstupy analýzy tvoří další velký celek aplikace, nicméně z důvodu omezení zadání této práce se jím více nebudu zabývat.

5.2.2.3 Detekce témat

V kapitole o implementaci dosud nepadla zmínka o detekci témat, o které vlastně celá tato práce je. Celý analyzátor ještě není hotový, ale to mi nebrání jej použít pro detekci témat. Celá síť je vyhodnocena kladně, pokud je nalezena minimálně jedna posloupnost správně vyhodnocených uzlů od počátku sítě až ke koncovému uzlu (uzel bez dalších následovníků). Pro získání výsledků za účelem porovnání s druhou metodou nadefinujeme síť tak, abychom tuto vlastnost splnili. Nadefinuji takovou posloupnost uzlů, která odpovídá jednotlivým tématům. Jelikož mám dvě témata, která se vzájemně vylučují, bude mi stačit nadefinovat pouze jednu síť. Pokud bude síť v nahrávce splněna, vím že tato nahrávka odpovídá tématu

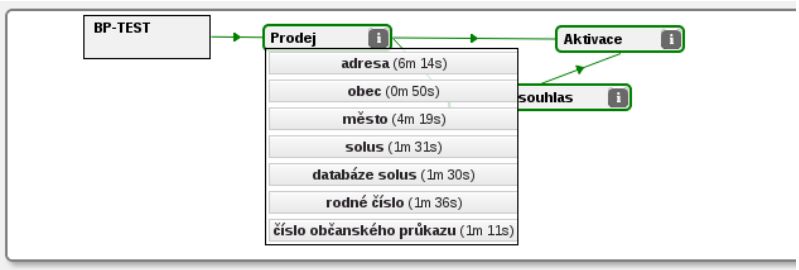
představovaném touto sítí, pokud síť není splněna, jedná se o téma druhé. Snadno si pak lze zjistit všechny výsledky a jednoduše spočítat úspěšnost detekce témat.

Na obrázku 5.2 vidíme síť detekující úspěšné hovory. Síť obsahuje tři uzly, které v nahrávce detekují místa typická pro úspěšné hovory.



Obrázek 5.2: Síť použitá pro detekci úspěšného hovoru.

Obrázek 5.3 ukazuje vyhodnocenou síť pro jednu nahrávku. Splněné uzly jsou zelené, nesplněné jsou ohraničeny červeně. Na obrázku lze také vidět seznam detekovaných slov pro první uzel. Seznam zobrazuje jednotlivá klíčová slova společně s časy výskytu v nahrávce.



Obrázek 5.3: Ukázka vyhodnocení sítě pro jednu nahrávku.

5.3 Řešení pomocí LVCSR

Původně měla být místo KWS použita technologie LVCSR, kde by se výsledný přepis nahrávek indexoval pomocí vhodného nástroje. Pro potřeby ukládání a fulltextového vyhledávání byl zvolen nástroj *Elasticsearch*[1].

5.3.1 Návrh řešení

1. zpracuj nahrávky pomocí LVCSR a přepis ulož pomocí nástroje Elasticsearch
2. sestav síť pro analýzu nahrávek v grafickém rozhraní
3. vyhodnoť scénář nad nahrávkami
   (a) pokud přepis odpovídá scénáři, našli jsme téma
   (b) pokud ne, téma jsme nenašli

[1] http://www.elasticsearch.org/

4. ulož výsledky do databáze

5.3.2 Elasticsearch

Tento nástroj byl vybrán hned z několika důvodů. Nástroj je k dispozici zdarma, má širokou komunitu, dobré reference od velkých celosvětově známých firem (The Guardian, GitHub, ...), dobrou dokumentaci, nabízí další nástroje pro vizualizaci, které by se mohly v budoucnu využívat, snadná konfigurace pro jazyky a v neposlední řadě nabízí API rozhraní v Javě.

Elasticsearch je komplexní nástroj zaměřený v prví řadě na fulltextové vyhledávání, dále ovšem podporuje kladení dotazů za účelem dostání konkrétních výsledků či statistik. Je postaven nad knihovnou *Lucene* [2]. Lucene je výkonná open source knihovna pro fulltextové vyhledávání napsána v jazyce Java. Lucene poskytuje nízkoúrovňové API, které je zapouzdřeno v nástroji Elasticsearch.

Jednotlivé **dokumenty**, v našem případě přepisy nahrávek, jsou ukládány v **indexech**. Procesu ukládání dokumentů se říká indexování. Každý dokument je specifikován **typem** a skládá se z jednotlivých **polí**. Analogii s relační databází ilustruje následující tabulka.

Relační databáze	Databáze	Tabulka	Řádek	Sloupec
Elasticsearch	Index	Typ	Dokument	Pole

5.3.2.1 Konfigurace Elasticsearch

Nejdříve je nutné nakonfigurovat Elasticsearch pro češtinu, protože každý jazyk má svá specifika, svoji slovní zásobu atd. Dříve než je dokument zaindexován, projde vstupní analýzou, ta se sestává z řady filtrů. Účelem těchto filtrů je předpřipravit vstupní text, například odstranit slova bez informační hodnoty (předložky, spojky, ...), převést slova na základní tvar (ořezat předpony a přípony, převést slovo do 1. pádu, ...). Konfigurace češtiny je v příloze B.1, při konfiguraci češtiny jsem vycházel z článku [13]. Konfigurační soubor je umístěn v souboru *config/elasticsearch.yml* .

Dále je nutné nakonfigurovat typy dokumentů, v našem případě definujeme typ *nahrávka*. Konfigurace je v příloze B.2. Konfigurace je v jazyce *json* a udává strukturu dokumentu. Soubor s definicí typu je umístěn v *config/mappings/_default/record.json*. Typ nahrávka představuje strukturu nahrávky hovoru. Každá nahrávka patří do kampaně (idSource), volá ji operátor (idOperator), má svůj čas vytvoření (createdTime), své unikátní ID (callId), dále se skládá z kanálů (1 kanál v případě mono, 2 kanály v případě stereo nahrávky). Každý kanál obsahuje list klíčových slov, která byla rozpoznána řečovým dekodérem (LVCSR), viz 2.5. Všechny položky mají nastaven atribut *not_analyzed*, který říká, daný údaj neanalyzuj a zaindexuj jej, tak jak je. Jediná položka *text* nesoucí informace o rozpoznaném textu má specifikovaný analyzátor, který jsme nastavili v předchozím odstavci.

5.3.3 Implementace

Při implementaci tohoto způsobu analýzy pomocí nástroje Elasticsearch jsem narazil na jeden vážný problém. Elasticsearch ve verzi 1.4 nepodporuje vnořené dotazy. Co to znamená, vysvětlím dále. Tento problém mi zatím neumožňuje dále pokračovat, ale příslib autorů

[2] http://lucene.apache.org/

na oficiálním fóru, že verze 1.5 již tuto funkcionalitu bude obsahovat, mě nenutí zatím použít jiný nástroj, ale vyčkat. Momentálně je situace taková, že verze 1.5 již vyšla, ale Java API není stále k dispozici.

5.3.3.1 Vnořené dotazování

V odstavci 5.3.2.1 defunuji typ Nahrávka, do tohoto typu je vnořen typ Kanál a do něj typ Klíčové slovo. V tuto chvíli nastává problém, kdy se potřebuji dotazovat na klíčová slova, která jsou ovšem vnořená. Položím dotaz, zda se konkrétní klíčové slovo nachází v nahrávce. Řekněme že ano, v tom případě potřebuji toto slovo získat, abych si mohl přečíst další údaje (čas výskytu, ...). Elasticsearch ve verzi 1.4 ovšem vrací celý kořenový dokument se všemi jeho vnořenými dokumenty, to jest se všemi slovy a ne pouze s tím, na které jsem se dotazoval.

5.3.4 Další vývoj

Momentálně je pro řečový analyzátor použita technologie KWS, nicméně v budoucnu dojde k přechodu na technologie LVCSR, která sice za cenu většího výpočetního výkonu nabízí řadu výhod. Tento přechod bude pouze obnášet změnu logické vrstvy aplikace, databázová vrstva pro ukládání výsledků zřejmě zůstane bez větších změn stejně tak jako uživatelské rozhraní. Tento fakt je zejména důležitý proto, že implementace s technologií KWS se nedělá zbytečně, ale že mnoho částí kódu bude možné znovu použít.

Kapitola 6

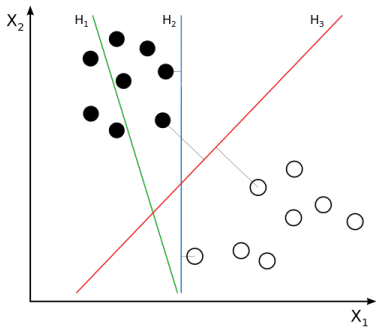
Detekce témat založená na strojovém učení

Strojové učení je vědecká disciplína zabývající se algoritmy, které mají schopnost učit se z dat. Pojem učení představuje nastavení vnitřního stavu algoritmu tak, aby nejlépe reflektoval svoje okolí. Obecně lze práci s touto třídou algoritmů rozdělit do dvou fází. První fází je trénink algoritmu, vstupem je trénovací sada dat. V této fázi dochází k onomu učení, výstupem je tzv. model, který je využíván v druhé fázi. Tato fáze představuje již samotné použití, které zahrnuje širokou řadu úloh, mezi které se řadí klasifikace, která je v této práci použita. V našem případě algoritmus na základě modelu klasifikuje nahrávky do tříd podle tématu, které v nich zaznělo.

Strojové učení představuje velmi široké a pestré odvětví informatiky, proto se dále v této práci zaměřím pouze na metodu, kterou jsem použil pro klasifikaci.

6.1 SVM

***S**upport **V**ector **M**achine* je třída metod založených na učení s učitelem. Základní využití je klasifikace a regresní analýza. Hlavní princip spočívá v nalezení oddělovače (nadroviny) v hledaném prostoru, který nejlépe rozdělí odlišné třídy. Taková nadrovina musí zároveň rozdělovat prostor s největší možnou vzdáleností nejbližších bodů obou tříd. Tuto skutečnost ilustruje obrázek 6.1, kde nadrovina *H3* je ideální.



Obrázek 6.1: Příklad dvou tříd a tří nadrovin. Přebráno z [14]

Hlavní fází při budování modelu je výpočet nadroviny, při kterém algoritmus hledá mezní vektory (body) tzv. support vectors, které oddělují třídy. Základní algoritmus SVM uvažuje klasifikaci dvou tříd pomocí lineární nadroviny. Nicméně existují modifikace pro n-rovin a také nelineární klasifikátory. Ty obvykle dosahují lepší úspěšnosti než ty lineární, ale za cenu většího výpočetního výkonu. Pro lineárně neseparovatelná data se využívá tzv. jádrových transformací. Tyto data jsou převedena do vícedimenzionálního prostoru, kde již jsou lineárně separovatelná.

SVM klasifikátory se osvědčily jako velmi spolehlivé a robustní pro data, jejichž vektory obsahují vysoký počet dimenzí. Takovým typem dat je například text, v našem případě přepisy nahrávek. Další pozitivní vlastností je počet trénovacích dat, kdy stačí vcelku malé množství. Velikost učící a testovací sady dat, počet dimenzí vektoru, stejně tak použití klasifikátorů je popsáno v následující kapitole 6.2, více o SVM lze nalézt na [3], [12].

6.2 LIBSVM

Pro práci a experimentování s SVM jsem využil knihovnu *LIBSVM*[1], která nabízí lineární i nelineární klasifikátory, dále má implementované API v několika jazycích, testovací sady dat a užitečný návod pro začátečníky [5].

6.2.1 Příprava dat

Před samotným testováním je nutné připravit sadu dat. To znamená upravit data do formátu, který vyžaduje knihovna *LIBSVM*. Takto upravená data je pak ještě nutné rozdělit na dvě části - trénovací a testovací sadu. Trénovací sada slouží k vytvoření modelu (klasifikátoru) a pomocí testovací sady ověříme jeho úspěšnost.

Vstupní data mají podobu přepsaných nahrávek, zároveň ke každé nahrávce známe její téma. Přepis je nutné převést do vektorové podoby. Proto můžeme použít celé řady technik. Já jsem zvolil algoritmus *TF-IDF*[2]. Téma nahrávky je označeno číslem, moje testovací nahrávky mohou být právě v jednom ze dvou témat, proto jsem zvolil označení 0 nebo 1.

Tabulka 6.1: Formát vstupních dat pro knihovnu *LIBSVM*

obecně	téma	vektor			
konrétněji	téma	index_1 : dimenze_1	index_n : dimenze_n		
příklad	0	1:3.1	2:0.1	3:5.154	4:3.145

6.2.1.1 Převod do LIBSVM formátu

Jak jsem zmínil výše, vstupní přepsané nahrávky je nutné převést do formátu, kterému rozumí knihovna *LIBSVM*. Tato fáze obnáší dva kroky.

Prvním krokem je převod přepisu do vektorové podoby, pro který jsem použil algoritmus *TF-IDF*, princip fungování je popsán v [2]. Úkolem algoritmu je ohodnotit každé slovo v nahrávce a celou nahrávku uložit do vektorové podoby. Ohodnocení je číslo typu *double*. Velikost vektoru se odvíjí od velikosti slovníku (všech použitých slov). Druhý krok převede tato data na požadovaný tvar.

Pro tento úkol jsem napsal program v Javě. Vstupem programu je textový soubor ve formátu csv, kde na každém řádku je jedna nahrávka. Každý řádek začíná tématem (0 nebo 1), a následuje celý přepis nahrávky. Program ohodnotí každou nahrávku algoritmem *TF-IDF*. Použil jsem již naimplementovaný algortimus, který je ke stažení zde [8]. Zároveň z tohoto

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvm/
[2] http://en.wikipedia.org/wiki/Tf-idf

algoritmu získám celý slovník. Program poté vytvoří vektor s příslušným ohodnocením pro každou nahrávku. Výstupem programu je textový soubor v požadovaném formátu, viz 6.1.

6.2.1.2 Rozdělení dat

Předchozí program v Javě nám převedl vstupní data do správného formátu. Tyto data je nutné dále rozdělit na trénovací a testovací sadu. K tomuto úkolu nám slouží tři skripty - *getTestData.sh*, *checkId.sh* a *checkData.py*. První dva skripty jsem napsal v bashi a třetí je součástí *LIBSVM*.

getTestData.sh
Skript slouží k vytvoření testovací sady dat z celého setu, přebírá 3 parametry:

1. počet nahrávek z každého tématu
2. vstupní soubor s LIBSVM formátem z předchozího programu v Javě
3. výstupní soubor v tomtéž formátu s testovací sadou dat

Skript vyjme ze vstupního souboru požadovaný počet dat a zapíše je do výstupního souboru. To, co zůstane ve vstupním souboru, slouží jako trénovací sada.

checkId.sh
Úkolem skriptu je pouze ověřit, že vstupní soubor neobsahuje žádná duplicitní ID, což by mohlo ovlivnit správnost výsledků.

checkData.py
Skript ověří, zda jsou vstupní data (trénovací a testovací sada) validní, aby nedošlo během klasifikace k chybě.

6.2.2 Použití LIBSVM

Pro testovaní jsem použil skript *easy.py*, který je součástí *LIBSVM*. Skript přebírá dva parametry - trénovací a testovací sadu dat. Podle manuálu slouží pro začínající uživatele knihovny *LIBSVM*. To znamená, že ve skriptu je definována posloupnost úkonů, které je nutné vykonat, abychom dostali výsledky. Hlavní kroky jsou škálování dat, grid-search, trénování modelu a testování modelu. Skript volá příslušné funkce, které tyto procedury vykonají. Z toho také vyplývá, že jsou již veškeré parametry implicitně nastaveny.

Použil jsem tento skript kvůli výše jmenovaným důvodům, stačilo pouze v tomto skriptu měnit parametry jednotlivých kroků (interval škálování, typ kernelu pro fázi grid search a trénování modelu). Provedl jsem několik experimentů, zkoušel jsem jednak měnit parametry *LIBSVM* algoritmu, tak i různá nastavení *TF-IDF*. Výsledky jsou shrnuty v tabulce 6.2.

6.2.2.1 Fáze výpočtu

Škálování
Škálování, nebo také normalizace, je součástí přípravy dat a používá se k úpravě rozsahu dat. Data se škálují ze dvou hlavních důvodů:

1. rychlost výpočtu, data lépe konvergují
2. přesnější výsledky výpočtu, redukují se maxima a minima

Defaultní nastavení škálování v *LIBSVM* je v intervalu <-1,1>, jelikož algoritmus *TF-IDF* vrací pouze kladná čísla, změnil jsem škálovací interval <0,1>.

Grid search
Úkolem této fáze je stanovení nejlepších parametrů pro jádrové transformace, tak aby natrénovaný model byl v klasifikaci co nejúspěšnější. Během výpočtu je vstupní množina trénovacích sad rozdělena na stejně velké podmnožiny. Vždy se vybere jedna a na té se testuje, na zbylých se trénuje model. Tento postup se iterativně opakuje přes všechny množiny,jedná se o tzv. cross - validaci. Výhodou je díky nezávislosti dat možnost paralelního běhu, čímž se zkrátí doba výpočtu. Ve všech testech jsem spouštěl cross-validaci na 3 jádrech.

Trénování
Po stanovení nejlepších parametrů pro použitý kernel následuje samotné trénování. Výstupem je tzv. model, který slouží ke klasifikaci nových dat.

Testování
V této fázi probíhá testování úspěšnosti modelu. Tomu předchází ještě škálování testovacích dat. Použil jsem stejný interval jako při škálování dat trénovacích. Výstupem je úspěšnost s jakou model klasifikoval testovací data.

6.2.2.2 Experimentování a výsledky

Tabulka 6.2: Experimentování s *LIBSVM*

train [ks]	**test [ks]**	**úspěšnost [%]**	**čas**	**nastavení**
265	40	90.0	0m 3.8s	TF raw, linear, no grid, c=1
265	40	90.0	0m 19.9s	TF raw, linear, grid, c=0.03125
265	40	92.5	2m 29.5s	TF raw, RBF kernel, grid, c=8.0, g=0.00048828125
1000	318	88.2	0m 21.5s	TF raw, linear, no grid, c=1
1000	318	85.5	0m 22.3s	TF raw, linear, no grid, c=0.5
1000	318	88.7	2m 6.8s	TF raw, linear kernel, grid, c=0.03125
1000	318	88.7	22m 49.5s	TF raw, RBF kernel, grid, c=512.0, g=3.0517578125e-05
1000	318	89.3	2m 22.9s	TF log, linear, grid, c=0.03125
1000	318	90.6	21m 50.6s	TF log, RBF kernel, grid, c=32.0, g=0.0001220703125

Ve všech experimentech je použit interval škálování <0,1>, grid search byl spouštěn na 3 jádrech. Při testování jsem pouze měnil velikost sady dat, použil jsem dvě metody

algoritmu TF. Dále jsem používal pouze lineární a jedno nelineární (RBF) jádro. Celkem jsem provedl 9 testů. Čas jednotlivých testů byl měřen unixovým nástroje *time* a konkrétně se jedná o výstup *real time*.

Z výsledků vyplývá několik závěrů. Potvrzuje se, že použití nelineárního kernelu je více časově náročné. Zároveň pro můj typ dat nepřineslo výrazně lepší výsledky. První tři testy ukázaly, že i při malé sadě dat lze dosáhnout dobrých výsledků. Logaritmická normalizace algoritmu TF podávala lepší výsledky než jeho prostá forma. Také použití grid search pro stanovení nejlepších parametrů se osvědčilo. Delší doba výpočtu je vyvážena lepšími výsledky klasifikátoru.

Kapitola 7

Testovací sada dat a srovnání výsledků

7.0.3 Specifikace sady

Celá použitá testovací sada dat obsahuje 1318 nahrávek. Každá nahrávka spadá právě do jednoho ze dvou témat. Nahrávky jsou ve formátu *wav*, tak aby bylo možné nad nimi spouštět řečové technologie.

Charakter nahrávek je takový, že nahrávky představují hovory, v kterých se jedna strana snaží prodat produkt straně druhé. Prodej je buď úspěšný či naopak, právě tato skutečnost udává téma každé nahrávky.

Jelikož testovací sada obsahuje osobní údaje, které podléhají zákonu **č. 101/2000 Sb., o ochraně osobních údajů**, tak tato sada není veřejně k dispozici. Dostupná je pouze její verze v *LIBSVM* formátu, ze které již žádné osobní údaje nelze vyčíst.

7.0.4 Srovnání metod

Testy obou metod byly prováděny nad stejnou sadou dat čítající 318 nahrávek. Metoda založená na expertním přístupu kladení velmi přesného dotazu byla méně úspěšná. Úspěšnost by se dala zvýšit naposloucháním většího množství nahrávek a sestavením lepšího dotazu. Stejně tak úspěšnost přístupu založeném na strojovém učení by se dala zvýšit, například větší trénovací sadou, či nastavením lepších parametrů.

Vzhledem k tomu, že testované nahrávky občas obsahovaly nekonzistence. Zejména takové, kdy hovor nedopadl jednoznačně úspěšně, ale pouze s příslibem budoucího rozmyšlení. Takové hovory nebyly vždy označeny stejným tématem. I přes to všechno dopadly detekce témat obou metod poměrně dobře, viz následující tabulka 7.1.

Tabulka 7.1: Nejlepší výsledky obou porovnávaných metod.

Kladení velmi přesného dotazu [%]	**Strojové učení** [%]
82	90.6

Kapitola 8

Závěr

Předmětem práce byla detekce témat z mluvené řeči založená na dvou přístupech. Celé detekci témat předcházel převod mluvených nahrávek do textové podoby, kdy jsem použil řečové technologie společnosti Phonexia, konkrétně dvě - KWS (detekce klíčových slov) a LVCSR (přepis nahrávky do textu). První část práce se zabývá principem fungování těchto technologií.

První metoda, **expertní přístup kladení velmi přesného dotazu**, byla implementována v rámci aplikace SPAS společnosti Phonexia. Detekce témat je implementována jako součást řečového analyzátoru. Před samotnou detekcí je nutné sestavit dotaz, který zde má podobu sítě. Síť se buduje v grafickém editoru a je složena z uzlů a hran. Jednotlivé uzly představují dílčí analýzy hovoru, hrany jsou pak logické souvislosti mezi jednotlivými uzly. Pokud dojde ke správnému vyhodnocení sítě (je nalezena posloupnost úspěšně vyhodnocených uzlů od kořene až k listu), pak máme nalezeno hledané téma. Pokud výsledek sítě je záporný, téma se v dané nahrávce nevyskytuje. Řečový analyzátor ještě není zcela dokončen. Stále na něm pracuji, avšak současná podoba mi bez problému posloužila k detekci témat. Ač tato metoda dopadla, co se týče úspěšnosti, hůře, její použití je opodstatněné zejména z toho důvodu, že drtivá většina hovorů se odehrává podle předem sestaveného scénáře, který lze pomocí řečového analyzátoru v aplikaci SPAS reprezentovat sítí. Touto sítí lze rozpoznávat témata, ale hlavně lze kontrolovat části hovoru - povinné/zakázané fráze, což je funkcionalita, která je v kontaktních centrech velmi využívána.

Druhá varianta je založen na zcela odlišném přístupu, kdy je k rozpoznávání témat použit statistický klasifikátor. Pro natrénování klasifikátoru je použita **metoda strojového učení**. Na rozdíl od první metody zde odpadá lidský faktor (expert sestavující dotaz v podobě grafické sítě). Klasifikátor je natrénován na sadě trénovacích dat, poté je již připraven k rozpoznávání. Jako metodu strojového učení jsem použil SVM, konkrétně implementaci LIBSVM. Úspěšnost této metody byla zhruba o 10% lepší než metoda první.

Další směřování práce se týká výhradně prvního přístupu, jehož funkcionalita je neustále vyvíjena. Nejbližší kroky vývoje se budou týkat lepšího uživatelského rozhraní a zapracování detekce témat. Řečový analyzátor je víceúčelový a bude sloužit k analýze hovorů. To znamená ke kontrole kritických míst, dále k detekci témat a v konečné fázi i k dolování dat. Všechny tyto funkce budou muset být dobře zapouzdřeny, aby je mohl uživatel bez větších obtíží a zmatků používat.

Literatura

[1] BSAPI Documentation. [Online].
URL http://www.phonexia.cz/docs/bsapi/index.html

[2] Tf-Idf. [Online].
URL http://en.wikipedia.org/wiki/Tf%E2%80%93idf

[3] Aggarwal, C. C.; ChengXiang, Z.: *Mining Text Data*. Springer, 2012, ISBN 978-1-4614-3222-7.

[4] Chalupníček, K.: *Rozpoznávání diktované řeči pro medicínské aplikace*. Diplomová práce, Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních tecgnologií., Brno, 2004.

[5] Chih-Wei, H.; Chih-Chung, C.; Chih-Jen, L.: A Practical Guide to Support Vector Classification. 2010.
URL http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[6] Haderlein, T.; Nöth, E.: Teorie skrytých Markovových modelů. 2013, [Online], Presentation in course Interakce člověk - počítač v přirozeném jazyce (ICP). Západočeská univerzita V Plzni.
URL http://www.kiv.zcu.cz/studies/predmety/icp/ICP_LS13/icpkap05.pdf

[7] Hannemann, M.: Weighted Finite State Transducers in Automatic Speech Recognition. 10 2013, [Online], Presentation on ZRE lecture 10.04.2013.
URL http://www.fit.vutbr.cz/study/courses/ZRE/public/pred/10_wfst_lvcsr/zre_lecture_asr_wfst.pdf

[8] McNeill, W.: Tf-Idf implementace. [Online].
URL https://github.com/wpm/tfidf

[9] Moyal, A.; Aharonson, V.; Tetariy, E.; aj.: *Phonetic Search Methods for Large Speech Databases*. Springer, 2013, ISBN 978-1-4614-6488-4.

[10] Osmani, A.: Learning JavaScript Design Patterns. [Online].
URL http://addyosmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript

[11] Psutka, J.: *Mluvíme s počítačem česky*. Academia, 2006, ISBN 80-200-1309-1.

[12] Shih-Hung, W.: Support Vector Machine Tutorial.
URL http://www.csie.cyut.edu.tw/~shwu/PR_slide/SVM.pdf

[13] Vlček, L.: Elasticsearch: Vyhledáváme hezky česky. [Online].
URL http://www.zdrojak.cz/clanky/elasticsearch-vyhledavame-cesky/

[14] Weinberg, Z.: Svm separating hyperplanes. 2012.
URL http://en.wikipedia.org/wiki/File:
Svm_separating_hyperplanes_%28SVG%29.svg#file

[15] Zhezhela, O.: *Vizualizace výstupu z řečových technologií pro potřeby kontaktních center*. Diplomová práce, Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních tecgnologií., Brno, 2014.

Příloha A

Obsah CD

A.1 Zdrojové kódy

A.2 Elektronická verze této práce

Příloha B

Zdrojové kódy

B.1 Konfigurace češtiny pro Elasticsearch

```
index:
    analysis:
        analyzer:
            cs_hunspell:
                type: custom
                tokenizer: standard
                filter: [stopwords_CZ, cs_CZ, lowercase, stopwords_CZ, remove_duplicities]
        filter:
            stopwords_CZ:
                type: stop
                stopwords: [_czech_]
                ignore_case: true
            cs_CZ:
                type: hunspell
                locale: cs_CZ
                dedup: true
                recursion_level: 0
            remove_duplicities:
                type: unique
                only_on_same_position: true
```

B.2 Definice typu nahrávka pro Elasticsearch

```
{
    "record": { "properties": {
        "callId": { "type": "string", "index": "not_analyzed"},
        "createdTime": { "type": "date", "index": "not_analyzed"},
        "sellingCall": { "type": "boolean", "index": "not_analyzed"},
        "idOperator": { "type": "long", "index": "not_analyzed"},
        "idCalled": { "type": "long", "index": "not_analyzed"},
        "idSource": { "type": "long", "index": "not_analyzed"},
```

```
        "channelList": { "type": "nested", "properties": {
            "channelType": { "type": "long", "index": "not_analyzed"},

            "wordList": { "type": "nested", "properties": {
                "text": { "type": "string", "analyzer": "cs_hunspell"},
                "startTime": { "type": "double", "index": "not_analyzed"},
                "endTime": { "type": "double", "index": "not_analyzed"},
                "confidence": { "type": "double", "index": "not_analyzed"}
            }}
        }}
    }}
}
```